# 【Spring学习27】回顾总结Spring常用注解

作者：soonfly (/authorarticle.html?author=soonfly) 2017-04-10 ☆ 收录到我的专题 (/select.html?articleId=401742)

标签 注解 (http://www.bijishequ.com/info/search.html?searchText=注解) Bean (http://www.bijishequ.com/info/search.html?searchText=Bean) 配置 (http://www.bijishequ.com/info/search.html?searchText=配置) XML (http://www.bijishequ.com/info/search.html?searchText=XML) bean (http://www.bijishequ.com/info/search.html?searchText=bean)

前面已经多次用到了注解，如自动装配、扫描，bean初始化及销毁回调等
现在汇总并回顾一下:

## 1、 @Autowired 注解

要让@Autowired 起作用必须事先在 Spring 容器中声明 AutowiredAnnotationBeanPostProcessor Bean。

```
<!-- 该 BeanPostProcessor 将自动起作用，对标注 @Autowired 的 Bean 进行自动注入 -->
<bean class="org.springframework.beans.factory.annotation. AutowiredAnnotationBeanPostProcessor"/>
```

Spring 2.5 引入了 @Autowired 注解，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作:

```
package twm.spring;
import org.springframework.beans.factory.annotation.Autowired;

public class Test {
//下面是演示，正常开发中切记不要将几种方式作用在同一属性上。

//直接标注在属性上
@Autowired
private Sender sender;
private User user;

//标注在 Setter 方法上
@Autowired
public void setSender(Sender sender) {
this.sender = sender;
}
//@Autowired 注解标注在构造函数上
@Autowired
public Boss(Sender sender ,User user){
this.sender = sender;
this.user = user ;
}
//......
}
```

## 2、 @Qualifier 注解

当容器中存在同样类型的多个bean时，可以使用 @Qualifier 注解指定注入 Bean 的名称 :

```
@Autowired
public void setSender(@Qualifier("cellphoneSender")Sender sender) {
this.sender = sender;
}
```

@Autowired 和@Qualifier 结合使用时，自动注入的策略就从 byType 转变成 byName 了。注意：@Autowired 标注的是成员变量、方法以及构造函数，而@Qualifier 标注的是成员变量、方法入参、构造函数入参。

# 3、 @PostConstruct 和 @PreDestroy注解

在《Bean生命周期回调：初始化回调和销毁回调 》 (http://blog.csdn.net/soonfly/article/details/69484130)中已用到了这两个注解。
JSR-250 为初始化之后/销毁之前方法的指定定义了两个注解类，分别是 @PostConstruct 和 @PreDestroy，这两个注解只能应用于方法上。标注了 @PostConstruct 注解的方法将在类实例化后调用，而标注了 @PreDestroy 的方法将在类销毁之前调用。

```
package twm.spring;
import org.springframework.beans.factory.annotation.Autowired;

public class Test {
//......
@PostConstruct
public void myInit(){
System.out.println("postConstruct");
}

@PreDestroy
public void myDestory(){
System.out.println("preDestroy");
}
}
```

通过实现 InitializingBean/DisposableBean 接口，以及通过 <bean> 元素的init-method/destroy-method 属性进行配置，都只能为 Bean 指定一个初始化/销毁的方法。但是使用 @PostConstruct 和 @PreDestroy 注解却可以指定多个初始化 / 销毁方法，那些被标注 @PostConstruct 或 @PreDestroy 注解的方法都会在初始化 / 销毁时被执行。

# 4、@Resource

@Resource 的作用相当于 @Autowired，只不过 @Autowired 按 byType 自动注入，面@Resource 默认按 byName 自动注入。@Resource 有两个重要属性分别是 name 和 type。如果使用 name 属性，则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。都不写就默认按 byName 自动注入，如果找不到对应name的bean，则再按byType查找。

Resource 注解类位于 Spring 发布包的 lib/j2ee/common-annotations.jar 类包中，因此在使用之前必须将其加入到项目的类库中。

```
public class Test {
//下面是演示，正常开发中切记不要将几种方式作用在同一属性上。

@Resource(type="Sender.Class")
private Sender sender;

//标注在 Setter 方法上
@Resource(name="cellphoneSender")
public void setSender(Sender sender) {
this.sender = sender;
}
}
```

> 注：@Resource、@PostConstruct 以及
> @PreDestroy。它们不属于spring框架，而是在java中定义的。只不过spring框架支持这种java注解。

# 5、@Component

《自动检测扫描Bean》 (http://blog.csdn.net/soonfly/article/details/69359005)已经提到了通过注解可以自动扫描bean，并注册到容器中去，这样不需在XML 文件中通过 进行定义，从而减小XML体积，便于维护。 Spring提供了这样的注解：
**@Component** 表示一个组件 (Bean) ，可以作用在任何层次。
**@Service** 通常作用在业务层，功能与 @Component 相同。
**@Controller** 通常作用在控制层，功能与 @Component 相同。
**@Repository** 通常作用在数据持久层，功能与 @Component 相同。
通过在类上使用 @Repository、@Component、@Service 和 @Constroller 注解，Spring 会自动创建相应的 BeanDefinition 对象，并注册到 ApplicationContext 中。这些类就成了 Spring 受管组件。
例：

```
@Scope("prototype")
@Component(name="cellphoneSender")
public class Sender {
//.....实现
}
```

使用自动检测后扫描到的类，容器会将类名且第一个字母小写来作为bean的id。@Component 也提供了一个可选的入参name，用于指定 Bean 的名称。不过一般可不必指定，因为大多时侯都使用byType方式注入的。
默认情况下通过 @Component 定义的 Bean 都是 singleton 的，如果需要使用其它作用范围的 Bean，可以通过@Scope 注解来达到目标。

在使用 @Component 注解后，Spring 容器要启用类扫描机制以启用注解驱动 Bean 定义和注解驱动 Bean 自动注入的策略。Spring对 context 命名空间进行了扩展，提供了这一功能，请看下面的配置：

```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:c="http://www.springframework.org/schema/c" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd">
<context:component-scan base-package="twm.spring.start" />
</beans>
```

这里，所有通过 <bean> 元素定义 Bean 的配置内容已经被移除，仅需要添加一行 <context:component-scan/> 配置就解决所有问题了。 <context:component-scan/> 的 base-package 属性指定了需要扫描的类包，类包及其递归子包中所有的类都会被处理。

# 6、@Required注解

@Required注解应用于bean属性的setter方法，它表明影响的bean属性在配置时必须放在XML配置文件中。

```
@Required
public void setSender(Sender sender) {
this.sender = sender;
}
```

这样如果bean定义中没有给sender传值，就会报错。

```
<bean id="order" class="twm.spring.start.Order" autowire="byType">
<!-- <property name="sender" ref="cellphonesender"></property> -->
</bean>
```

比如把 <property name="sender" ref="cellphonesender"></property> 这句注释掉就会报错。

# 7、配置

值得注意的是:
当我们需要使用@Autowired注解，必须事先在Spring容器中声明AutowiredAnnotationBeanPostProcessor的Bean：

```
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor "/>
```

使用 @Required注解，就必须声明RequiredAnnotationBeanPostProcessor的Bean：

```
<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>
```

类似地，使用@Resource、@PostConstruct、@PreDestroy等注解就必须声明 CommonAnnotationBeanPostProcessor；使用@PersistenceContext注解，就必须声明 PersistenceAnnotationBeanPostProcessor的Bean。

这样的声明太不优雅，而Spring为我们提供了一种极为方便注册这些BeanPostProcessor的方式，即使用 <context:annotation- config/> 隐式地向 Spring容器注册AutowiredAnnotationBeanPostProcessor、RequiredAnnotationBeanPostProcessor、CommonAnnotationBeanPostProcessor以及PersistenceAnnotationBeanPostProcessor这4个BeanPostProcessor。如下：

```
<context:annotation-config/>
```

在我们使用注解时一般都会配置扫描包路径选项：

```
<context:component-scan base-package="pack.pack"/>
```

该配置项其实也包含了自动注入上述processor的功能，因此当使用 <context:component-scan/> 后，即可将 <context:annotation-config/> 也省去。

# 总结：

**注解配置相对于 XML 配置具有很多的优势：**

1、它可以充分利用 Java 的反射机制获取类结构信息，这些信息可以有效减少配置的工作。
2、 如使用 JPA 注解配置 ORM 映射时，我们就不需要指定 PO 的属性名、类型等信息，如果关系表字段和 PO属性名、类型都一致，甚至无需编写任务属性映射信息——因为这些信息都可以通过 Java 反射机制获取。
3、注解和 Java代码位于一个文件中，而 XML 配置采用独立的配置文件，大多数配置信息在程序开发完成后都不会调整，如果配置信息和 Java代码放在一起，有助于增强程序的内聚性。如果采用独立的 XML配置文件，程序员在编写一个功能时，往往需要在程序文件和配置文件中不停切换，这种思维上的不连贯会降低开发效率。

**注解配置和 XML 配置的适用场合**

是否有了这些 IOC 注解，我们就可以完全摒除原来 XML 配置的方式呢？答案是否定的。有以下几点原因：

1、注解配置不一定在先天上优于 XML 配置。如果 Bean 的依赖关系是固定的，（如 Service 使用了哪几个 DAO 类），这种配置信息不会在部署时发生调整，那么注解配置优于 XML 配置；反之如果这种依赖关系会在部署时发生调整，XML 配置显然又优于注解配置，因为注解是对 Java 源代码的调整，您需要重新改写源代码并重新编译才可以实施调整。

2、如果 Bean不是自己编写的类（如 JdbcTemplate、SessionFactoryBean 等），注解配置将无法实施，此时 XML配置是唯一可用的方式。

3、注解配置往往是类级别的，而 XML 配置则可以表现得更加灵活。比如相比于 @Transaction事务注解，使用 aop/tx 命名空间的事务配置更加灵活和简单。

所以在实现应用中，我们往往需要同时使用注解配置和 XML配置，对于类级别且不会发生变动的配置可以优先考虑注解配置；而对于那些第三方类以及容易发生调整的配置则应优先考虑使用 XML配置。Spring 会在具体实施 Bean 创建和 Bean 注入之前将这两种配置方式的元信息融合在一起。



原文链接：http://blog.csdn.net/soonfly/article/details/70023707



**相关标签**

Struts (/tag/list?tagId=1440) | Ioc (/tag/list?tagId=1441) | Spring (/tag/list?tagId=1438) | SpringMVC (/tag/list?tagId=1560) | 反射 (/tag/list?tagId=2452) | Collection (/tag/list?tagId=2453) | 动态代理 (/tag/list?tagId=2450) | 注解 (/tag/list?tagId=2451) | Servlet (/tag/list?tagId=2449) | Web服务 (/tag/list?tagId=1619) | ATOM (/tag/list?tagId=1623) | Xml (/tag/list?tagId=1621) | IWorkbench (/tag/list?tagId=519) | EnumMap (/tag/list?tagId=518) | HttpUriRequest (/tag/list?tagId=516) | StringUtils (/tag/list?tagId=515) | RSS (/tag/list?tagId=1622) | SOAP (/tag/list?tagId=1620) | BeanPostProcessor (/tag/list?tagId=1462) | RequestHandleEvent (/tag/list?tagId=1461) | ContextStoppedEvent (/tag/list?tagId=1460) | InitialingBean (/tag/list?tagId=1463) | DisposableBean (/tag/list?tagId=1464) | @Qualifier (/tag/list?tagId=1452) | @repository (/tag/list?tagId=1448) | @Autowired (/tag/list?tagId=1450) | @Resource (/tag/list?tagId=1451) | @component (/tag/list?tagId=1449) | J2EE (/tag/list?tagId=2397) | 编程语言 (/tag/list?tagId=2761) | java (/tag/list?tagId=2396) | web应用开发 (/tag/list?tagId=1595) | web应用 (/tag/list?tagId=1593) | we框架 (/tag/list?tagId=1594) | ApplicationListener (/tag/list?tagId=1456) | CGLib (/tag/list?tagId=1445) | Spring注解 (/tag/list?tagId=1446)

**相关文章**

Spring源码分析之AOP解析 (/detail/577731?p=)

Spring事务管理入门与进阶 (/detail/577593?p=)